This repository Search Pull requests Issues Gist

jlizier / jidt Unwatch

Code Issues 25 Pull requests 0 Projects 0 Wiki Pulse Graphs

PythonExamples

Joseph Lizier edited this page 5 days ago · 5 revisions

Examples of using the toolkit in Python

Demos > Python code examples

Python code examples

This page describes a basic set of demonstration scripts for using the toolkit in Python. The .py files can be found at demos/python in the svn or main distributions. We plan to have other more complicated examples available from the main Demos page in future.

Please see UseInPython for instructions on how to begin using the java toolkit from inside python.

Note that these examples use JPype -- you will need to alter them if you want to use another Python-Java interface.

This page contains the following code examples:

Example 1 - Transfer entropy on binary data

example1TeBinaryData.py - Simple transfer entropy (TE) calculation on binary data using the discrete TE calculator:

```
import jpype
import random
import numpy

# Change location of jar to match yours:
jarLocation = "../../infodynamics.jar"
# Start the JVM (add the "-Xmx" option with say 1024M if you get crashes due to not
jpype.startJVM(jpype.getDefaultJVMPath(), "-ea", "-Djava.class.path=" + jarLocation)

# Generate some random binary data.
sourceArray = [random.randint(0,1) for r in range(100)]
destArray = [0] + sourceArray[0:99]
sourceArray2 = [random.randint(0,1) for r in range(100)]

# Create a TE calculator and run it:
teCalcClass = jpype.JPackage("infodynamics.measures.discrete").TransferEntropyCalcul
teCalc = teCalcClass(2,1)
teCalc.initialise()

# First use simple arrays of ints, which we can directly pass in:
teCalc.addObservations(sourceArray, destArray)
print("For copied source, result should be close to 1 bit : %.4f" % teCalc.computeAv
teCalc.initialise()
teCalc.addObservations(sourceArray2, destArray)
print("For random source, result should be close to 0 bits: %.4f" % teCalc.computeAv

# Next, demonstrate how to do this with a numpy array
teCalc.initialise()
# Create the numpy arrays:
sourceNumpy = numpy.array(sourceArray, dtype=numpy.int)
destNumpy = numpy.array(destArray, dtype=numpy.int)
# The above can be passed straight through to JIDT in python 2:
# teCalc.addObservations(sourceNumpy, destNumpy)
# But you need to do this in python 3:
sourceNumpyJArray = jpype.JArray(jpype.JInt, 1)(sourceNumpy.tolist())
destNumpyJArray = jpype.JArray(jpype.JInt, 1)(destNumpy.tolist())
teCalc.addObservations(sourceNumpyJArray, destNumpyJArray)
print("Using numpy array for copied source, result confirmed as: %.4f" % teCalc.comp

jpype.shutdownJVM()
```

Example 2 - Transfer entropy on multidimensional binary data

example2TeMultidimBinaryData.py - Simple transfer entropy (TE) calculation on multidimensional binary data using the discrete TE calculator.

This example is important for Python JPype users, because it shows how to handle multidimensional arrays from Python to Java.

```
from jpype import *
import random

# Change location of jar to match yours:
jarLocation = "../../infodynamics.jar"
# Start the JVM (add the "-Xmx" option with say 1024M if you get crashes due to not
startJVM(getDefaultJVMPath(), "-ea", "-Djava.class.path=" + jarLocation)

# Create many columns in a multidimensional array, e.g. for fully random values:
# twoDTimeSeriesOctave = [[random.randint(0,1) for y in range(2)] for x in range(10)

# However here we want 2 rows by 100 columns where the next time step (row 2) is to
# value of the column on the left from the previous time step (row 1):
numObservations = 100
row1 = [random.randint(0,1) for r in range(numObservations)]
row2 = [row1[numObservations-1]] + row1[0:numObservations-1] # Copy the previous row
twoDTimeSeriesPython = []
twoDTimeSeriesPython.append(row1)
twoDTimeSeriesPython.append(row2)
twoDTimeSeriesJavaInt = JArray(JInt, 2)(twoDTimeSeriesPython) # 2 indicating 2D arra

# Create a TE calculator and run it:
teCalcClass = JPackage("infodynamics.measures.discrete").TransferEntropyCalculatorDi
teCalc = teCalcClass(2,1)
teCalc.initialise()
# Add observations of transfer across one cell to the right per time step:
teCalc.addObservations(twoDTimeSeriesJavaInt, 1)
result2D = teCalc.computeAverageLocalOfObservations()
print(('The result should be close to 1 bit here, since we are executing copy ' + \
      'operations of what is effectively a random bit to each cell here: %.3f ' + \
      'bits from %d observations') % (result2D, teCalc.getNumObservations()))
```

Example 3 - Transfer entropy on continuous data using kernel estimators

example3TeContinuousDataKernel.py - Simple transfer entropy (TE) calculation on continuous-valued data using the (box) kernel-estimator TE calculator.

```
from jpype import *
import random
import math

# Change location of jar to match yours:
jarLocation = "../../infodynamics.jar"
# Start the JVM (add the "-Xmx" option with say 1024M if you get crashes due to not
startJVM(getDefaultJVMPath(), "-ea", "-Djava.class.path=" + jarLocation)

# Generate some random normalised data.
```

```
numObservations = 1000
covariance=0.4
# Source array of random normals:
sourceArray = [random.normalvariate(0,1) for r in range(numObservations)]
# Destination array of random normals with partial correlation to previous value of
destArray = [0] + [sum(pair) for pair in zip([covariance*y for y in sourceArray[0:nu
                                              [(1-covariance)*y for y in [random.norm
# Uncorrelated source array:
sourceArray2 = [random.normalvariate(0,1) for r in range(numObservations)]
# Create a TE calculator and run it:
teCalcClass = JPackage("infodynamics.measures.continuous.kernel").TransferEntropyCal
teCalc = teCalcClass()
teCalc.setProperty("NORMALISE", "true") # Normalise the individual variables
teCalc.initialise(1, 0.5) # Use history length 1 (Schreiber k=1), kernel width of 0.
teCalc.setObservations(JArray(JDouble, 1)(sourceArray), JArray(JDouble, 1)(destArray
# For copied source, should give something close to 1 bit:
result = teCalc.computeAverageLocalOfObservations()
print("TE result %.4f bits; expected to be close to %.4f bits for these correlated G
    (result, math.log(1/(1-math.pow(covariance,2)))/math.log(2)))
teCalc.initialise() # Initialise leaving the parameters the same
teCalc.setObservations(JArray(JDouble, 1)(sourceArray2), JArray(JDouble, 1)(destArra
# For random source, it should give something close to 0 bits
result2 = teCalc.computeAverageLocalOfObservations()
print("TE result %.4f bits; expected to be close to 0 bits for uncorrelated Gaussian
    result2)
```

Example 4 - Transfer entropy on continuous data using Kraskov estimators

example4TeContinuousDataKraskov.py - Simple transfer entropy (TE) calculation on continuous-valued data using the Kraskov-estimator TE calculator.

```
from jpype import *
import random
import math

# Change location of jar to match yours:
jarLocation = "../../infodynamics.jar"
# Start the JVM (add the "-Xmx" option with say 1024M if you get crashes due to not
startJVM(getDefaultJVMPath(), "-ea", "-Djava.class.path=" + jarLocation)

# Generate some random normalised data.
numObservations = 1000
covariance=0.4
# Source array of random normals:
sourceArray = [random.normalvariate(0,1) for r in range(numObservations)]
# Destination array of random normals with partial correlation to previous value of
destArray = [0] + [sum(pair) for pair in zip([covariance*y for y in sourceArray[0:nu
                                              [(1-covariance)*y for y in [random.norm
```

```
# Uncorrelated source array:
sourceArray2 = [random.normalvariate(0,1) for r in range(numObservations)]
# Create a TE calculator and run it:
teCalcClass = JPackage("infodynamics.measures.continuous.kraskov").TransferEntropyCa
teCalc = teCalcClass()
teCalc.setProperty("NORMALISE", "true") # Normalise the individual variables
teCalc.initialise(1) # Use history length 1 (Schreiber k=1)
teCalc.setProperty("k", "4") # Use Kraskov parameter K=4 for 4 nearest points
# Perform calculation with correlated source:
teCalc.setObservations(JArray(JDouble, 1)(sourceArray), JArray(JDouble, 1)(destArray
result = teCalc.computeAverageLocalOfObservations()
# Note that the calculation is a random variable (because the generated
#  data is a set of random variables) - the result will be of the order
#  of what we expect, but not exactly equal to it; in fact, there will
#  be a large variance around it.
print("TE result %.4f nats; expected to be close to %.4f nats for these correlated G
    (result, math.log(1/(1-math.pow(covariance,2)))))
# Perform calculation with uncorrelated source:
teCalc.initialise() # Initialise leaving the parameters the same
teCalc.setObservations(JArray(JDouble, 1)(sourceArray2), JArray(JDouble, 1)(destArra
result2 = teCalc.computeAverageLocalOfObservations()
print("TE result %.4f nats; expected to be close to 0 nats for these uncorrelated Ga
```

Example 5 - Multivariate transfer entropy on binary data

example5TeBinaryMultivarTransfer.py - Multivariate transfer entropy (TE) calculation on binary data using the discrete TE calculator.

```
from jpype import *
import random
from operator import xor

# Change location of jar to match yours:
jarLocation = "../../infodynamics.jar"
# Start the JVM (add the "-Xmx" option with say 1024M if you get crashes due to not
startJVM(getDefaultJVMPath(), "-ea", "-Djava.class.path=" + jarLocation)

# Generate some random binary data.
numObservations = 100
sourceArray = [[random.randint(0,1) for y in range(2)] for x in range(numObservation
sourceArray2= [[random.randint(0,1) for y in range(2)] for x in range(numObservation
# Destination variable takes a copy of the first bit of the source in bit 1,
#  and an XOR of the two bits of the source in bit 2:
destArray = [[0, 0]]
for j in range(1,numObservations):
    destArray.append([sourceArray[j-1][0], xor(sourceArray[j-1][0], sourceArray[j-1]

# Create a TE calculator and run it:
teCalcClass = JPackage("infodynamics.measures.discrete").TransferEntropyCalculatorDi
teCalc = teCalcClass(4,1)
```

```
teCalc.initialise()
# We need to construct the joint values of the dest and source before we pass them i
#  and need to use the matrix conversion routine when calling from Matlab/Octave:
mUtils= JPackage('infodynamics.utils').MatrixUtils
teCalc.addObservations(mUtils.computeCombinedValues(sourceArray, 2), \
        mUtils.computeCombinedValues(destArray, 2))
result = teCalc.computeAverageLocalOfObservations()
print('For source which the 2 bits are determined from, result should be close to 2
teCalc.initialise()
teCalc.addObservations(mUtils.computeCombinedValues(sourceArray2, 2), \
        mUtils.computeCombinedValues(destArray, 2))
result2 = teCalc.computeAverageLocalOfObservations()
print('For random source, result should be close to 0 bits in theory: %.3f' % result
print('The result for random source is inflated towards 0.3 due to finite observatio
```

Example 6 - Dynamic dispatch with Mutual info calculator

example6DynamicCallingMutualInfo.py - This example shows how to write Python code to take advantage of the common interfaces defined for various information-theoretic calculators. Here, we use the common form of the `infodynamics.measures.continuous.MutualInfoCalculatorMultiVariate` interface (which is never named here) to write common code into which we can plug one of three concrete implementations (kernel estimator, Kraskov estimator or linear-Gaussian estimator) by dynamically supplying the class name of the concrete implementation.

Note -- users of the v1.0 distribution will need to separately download the readFloatsFile.py module, which was accidentally not included in this release.

```
from jpype import *
import random
import string
import numpy
import readFloatsFile

# Change location of jar to match yours:
jarLocation = "../../infodynamics.jar"
# Start the JVM (add the "-Xmx" option with say 1024M if you get crashes due to not
startJVM(getDefaultJVMPath(), "-ea", "-Djava.class.path=" + jarLocation)

#---------------------
# 1. Properties for the calculation (these are dynamically changeable):
# The name of the data file (relative to this directory)
datafile = '../data/4ColsPairedNoisyDependence-1.txt'
# List of column numbers for univariate time seres 1 and 2:
#  (you can select any columns you wish to be contained in each variable)
univariateSeries1Column = 0 # array indices start from 0 in python
univariateSeries2Column = 2
# List of column numbers for joint variables 1 and 2:
```

```
#  (you can select any columns you wish to be contained in each variable)
jointVariable1Columns = [0,1] # array indices start from 0 in python
jointVariable2Columns = [2,3]
# The name of the concrete implementation of the interface
#  infodynamics.measures.continuous.MutualInfoCalculatorMultiVariate
#  which we wish to use for the calculation.
# Note that one could use any of the following calculators (try them all!):
#  implementingClass = "infodynamics.measures.continuous.kraskov.MutualInfoCalculato
#  implementingClass = "infodynamics.measures.continuous.kernel.MutualInfoCalculator
#  implementingClass = "infodynamics.measures.continuous.gaussian.MutualInfoCalculat
implementingClass = "infodynamics.measures.continuous.kraskov.MutualInfoCalculatorMu

#---------------------
# 2. Load in the data
data = readFloatsFile.readFloatsFile(datafile)
# As numpy array:
A = numpy.array(data)
# Pull out the columns from the data set for a univariate MI calculation:
univariateSeries1 = A[:,univariateSeries1Column]
univariateSeries2 = A[:,univariateSeries2Column]
# Pull out the columns from the data set for a multivariate MI calculation:
jointVariable1 = A[:,jointVariable1Columns]
jointVariable2 = A[:,jointVariable2Columns]

#--------------------
# 3. Dynamically instantiate an object of the given class:
# (in fact, all java object creation in python is dynamic - it has to be,
#  since the languages are interpreted. This makes our life slightly easier at this
#  point than it is in demos/java/example6 where we have to handle this manually)
indexOfLastDot = string.rfind(implementingClass, ".")
implementingPackage = implementingClass[:indexOfLastDot]
implementingBaseName = implementingClass[indexOfLastDot+1:]
miCalcClass = eval('JPackage(\'%s\').%s' % (implementingPackage, implementingBaseNam
miCalc = miCalcClass()

#--------------------
# 4. Start using the MI calculator, paying attention to only
#  call common methods defined in the interface type
#  infodynamics.measures.continuous.MutualInfoCalculatorMultiVariate
#  not methods only defined in a given implementation class.
# a. Initialise the calculator for a univariate calculation:
miCalc.initialise(1, 1)
# b. Supply the observations to compute the PDFs from:
miCalc.setObservations(univariateSeries1, univariateSeries2)
# c. Make the MI calculation:
miUnivariateValue = miCalc.computeAverageLocalOfObservations()

#---------------------
# 5. Continue onto a multivariate calculation, still
#    only calling common methods defined in the interface type.
# a. Initialise the calculator for a multivariate calculation
#   to use the required number of dimensions for each variable:
```

```
miCalc.initialise(len(jointVariable1Columns), len(jointVariable2Columns))
# b. Supply the observations to compute the PDFs from:
miCalc.setObservations(jointVariable1, jointVariable2)
# c. Make the MI calculation:
miJointValue = miCalc.computeAverageLocalOfObservations()

print("MI calculator %s computed the univariate MI(%d;%d) as %.5f and joint MI([%s];
    (implementingClass, univariateSeries1Column, univariateSeries2Column, miUnivaria
    str(jointVariable1Columns).strip('[]'), str(jointVariable2Columns).strip('[]'),
```

Example 7 - Ensemble method with transfer entropy on continuous data using Kraskov estimators

example7EnsembleMethodTeContinuousDataKraskov.py - This example shows calculation of transfer entropy (TE) by supplying an ensemble of samples from multiple time series. We use continuous-valued data using the Kraskov-estimator TE calculator here. We also demonstrated local TE calculation in this case. The py file will be available in distributions from v1.4.

```
from jpype import *
import random
import math

# Change location of jar to match yours:
jarLocation = "../../infodynamics.jar"
# Start the JVM (add the "-Xmx" option with say 1024M if you get crashes due to not
startJVM(getDefaultJVMPath(), "-ea", "-Djava.class.path=" + jarLocation)

# Generate some random normalised data.
numObservations = 1000
covariance=0.4
numTrials=10
kHistoryLength=1

# Create a TE calculator and run it:
teCalcClass = JPackage("infodynamics.measures.continuous.kraskov").TransferEntropyCa
teCalc = teCalcClass()
teCalc.setProperty("k", "4") # Use Kraskov parameter K=4 for 4 nearest points
teCalc.initialise(kHistoryLength) # Use target history length of kHistoryLength (Sch
teCalc.startAddObservations()

for trial in range(0,numTrials):
    # Create a new trial, with destArray correlated to
    #  previous value of sourceArray:
    sourceArray = [random.normalvariate(0,1) for r in range(numObservations)]
    destArray = [0] + [sum(pair) for pair in zip([covariance*y for y in sourceArray[
        [(1-covariance)*y for y in [random.normalvariate(0,1) for r in range(numObse

    # Add observations for this trial:
    print("Adding samples from trial %d ..." % trial)
```

```
        teCalc.addObservations(JArray(JDouble, 1)(sourceArray), JArray(JDouble, 1)(destA

    # We've finished adding trials:
    print("Finished adding trials")
    teCalc.finaliseAddObservations()

    # Compute the result:
    print("Computing TE ...")
    result = teCalc.computeAverageLocalOfObservations()
    # Note that the calculation is a random variable (because the generated
    #  data is a set of random variables) - the result will be of the order
    #  of what we expect, but not exactly equal to it; in fact, there will
    #  be some variance around it (smaller than example 4 since we have more samples).
    print("TE result %.4f nats; expected to be close to %.4f nats for these correlated G
        (result, math.log(1.0/(1-math.pow(covariance,2)))))

    # And here's how to pull the local TEs out corresponding to each input time series.
    # Normally you would need to track how to split these up yourself -- here
    # it's easy because our input time series are all of the same length
    localTEs=teCalc.computeLocalOfPreviousObservations()
    localValuesPerTrial = int(len(localTEs)/numTrials)  # Need to convert to int for ind
    for trial in range(0,numTrials):
        startIndex = localValuesPerTrial*trial
        endIndex = localValuesPerTrial*(trial+1)-1
        print("Local TEs for trial %d go from array index %d to %d" % (trial, startIndex
        print("  corresponding to time points %d:%d (indexed from 0) of that trial" % (k
        # Access the local TEs for this trial as:
        localTEForThisTrial = localTEs[startIndex:endIndex]
```

Example 9 - Transfer entropy on continuous data using Kraskov estimators with auto-embedding

example9TeKraskovAutoEmbedding.py - This example shows how to make a Transfer entropy (TE) calculation on continuous-valued data using the Kraskov-estimator TE calculator, with automatic selection of embedding parameters (using the Ragwitz criterion). The py file will be available in distributions from v1.4.

Note -- users of the v1.0 distribution will need to separately download the readFloatsFile.py module, which was accidentally not included in this release.

```
from jpype import *
import random
import math
import numpy
import readFloatsFile

# Change location of jar to match yours:
jarLocation = "../../infodynamics.jar"
```

```
# Start the JVM (add the "-Xmx" option with say 1024M if you get crashes due to not
startJVM(getDefaultJVMPath(), "-ea", "-Djava.class.path=" + jarLocation)

# Examine the heart-breath interaction that Schreiber originally looked at:
datafile = '../data/SFI-heartRate_breathVol_bloodOx.txt'
data = readFloatsFile.readFloatsFile(datafile)
# As numpy array:
A = numpy.array(data)
# Select data points 2350:3550, pulling out the relevant columns:
breathRate = A[2350:3551,1];
heartRate = A[2350:3551,0];

# Create a Kraskov TE calculator:
teCalcClass = JPackage("infodynamics.measures.continuous.kraskov").TransferEntropyCa
teCalc = teCalcClass()

# Set properties for auto-embedding of both source and destination
#  using the Ragwitz criteria:
#  a. Auto-embedding method
teCalc.setProperty(teCalcClass.PROP_AUTO_EMBED_METHOD,
        teCalcClass.AUTO_EMBED_METHOD_RAGWITZ)
#  b. Search range for embedding dimension (k) and delay (tau)
teCalc.setProperty(teCalcClass.PROP_K_SEARCH_MAX, "6")
teCalc.setProperty(teCalcClass.PROP_TAU_SEARCH_MAX, "6")
# Since we're auto-embedding, no need to supply k, l, k_tau, l_tau here:
teCalc.initialise()
# Compute TE from breath (column 1) to heart (column 0)
teCalc.setObservations(breathRate, heartRate)
teBreathToHeart = teCalc.computeAverageLocalOfObservations()

# Check the auto-selected parameters and print out the result:
optimisedK = int(teCalc.getProperty(teCalcClass.K_PROP_NAME))
optimisedKTau = int(teCalc.getProperty(teCalcClass.K_TAU_PROP_NAME))
optimisedL = int(teCalc.getProperty(teCalcClass.L_PROP_NAME))
optimisedLTau = int(teCalc.getProperty(teCalcClass.L_TAU_PROP_NAME))
print(("TE(breath->heart) was %.3f nats for (heart embedding:) k=%d," + \
        "k_tau=%d, (breath embedding:) l=%d,l_tau=%d optimised via Ragwitz criteria"
        (teBreathToHeart, optimisedK, optimisedKTau, optimisedL, optimisedLTau))

# Next, embed the destination only using the Ragwitz criteria:
teCalc.setProperty(teCalcClass.PROP_AUTO_EMBED_METHOD,
        teCalcClass.AUTO_EMBED_METHOD_RAGWITZ_DEST_ONLY)
teCalc.setProperty(teCalcClass.PROP_K_SEARCH_MAX, "6")
teCalc.setProperty(teCalcClass.PROP_TAU_SEARCH_MAX, "6")
# Since we're only auto-embedding the destination, we supply
#  source embedding here (to overwrite the auto embeddings from above):
teCalc.setProperty(teCalcClass.L_PROP_NAME, "1")
teCalc.setProperty(teCalcClass.L_TAU_PROP_NAME, "1")
# Since we're auto-embedding, no need to supply k and k_tau here:
teCalc.initialise()
# Compute TE from breath (column 1) to heart (column 0)
teCalc.setObservations(breathRate, heartRate)
```

```
teBreathToHeartDestEmbedding = teCalc.computeAverageLocalOfObservations()

# Check the auto-selected parameters and print out the result:
optimisedK = int(teCalc.getProperty(teCalcClass.K_PROP_NAME))
optimisedKTau = int(teCalc.getProperty(teCalcClass.K_TAU_PROP_NAME))
print(("TE(breath->heart) was %.3f nats for (heart embedding:) k=%d," + \
        "k_tau=%d, optimised via Ragwitz criteria, plus (breath embedding:) l=1,l_ta
        (teBreathToHeartDestEmbedding, optimisedK, optimisedKTau))
```

JIDT -- Java Information Dynamics Toolkit -- Joseph Lizier *et al.*

© 2016 GitHub, Inc. Terms Privacy Security Status Help Contac